L41: Lab Setup

Dr Robert N. M. Watson Dr Graeme Jenkinson

Michaelmas Term 2017

L41 is taught through a blend of lectures and laboratory experiments. The purpose of the labs is threefold: to teach you about real-world operating systems, to teach you experimental methodology and practical skills, and to provide fodder for assessment. You will use tools such as DTrace to explore the behaviour of the system through 'potted' example programs that will trigger OS behaviours for you to investigate. Each lab is structured as a set of mandatory experimental questions; take care to ensure that your lab report addresses all of the assigned experimental questions.

Experimental platform

Our experimental platform is the open-source FreeBSD operating system running on the BeagleBone Black (BBB) board, described in the remainder of this handout.

The operating system: FreeBSD

We will be using the open-source FreeBSD operating system's ARMv7 port on the BeagleBone Black. FreeBSD is of particular interest due to its tight integration of a number of tracing and measurement tools (e.g., DTrace) and that it is built by default with the Clang/LLVM compiler suite, which make it easier to insert additional instrumentation. You can learn more about FreeBSD by visiting the FreeBSD Project's website:

`https://www.FreeBSD.org/`

The course text, *The Design and Implementation of the FreeBSD Operating System, Second Edition* will be a useful reference, covering concepts such as the process model, inter-process communications, and filesystems. There is also a section on the implementation of DTrace, which may be useful background material for the labs. *DTrace: Dynamic Tracing in Oracle Solaris, Mac OS X and FreeBSD* is a detailed, user-facing guide on how to use DTrace, and will also be a useful reference for the labs.

The board: BeagleBone Black

The BeagleBone Black board is based on a Texas Instruments System on Chip that employs a single-core ARM Cortex-A8 processor; it includes on-board flash, an Ethernet MAC, USB support, and an SD Card slot. This 32-bit processor is not the zippiest in the world – but it is widely deployed and fully functional with respect to many of the behaviours of interest for this course, including having processor performance counters. In our configuration, FreeBSD boots from the on-board SD Card.

We will access the board as a USB device from lab workstations or personal notebook computers. "USB target mode" allows the BBB to appear to be a set of USB devices to the attached workstation: an Ethernet device that can be used to SSH into the board, and a USB device that accesses a serial console. Power is also provided using the USB cable. These instructions describe how to access the board from a Linux workstation in the lab. Other configurations, such as Mac or Windows notebooks, or FreeBSD workstations, may also work with suitable adjustment to client-side tools.

While the BBB is not enormously expensive, they are often back ordered, so acquiring replacements can be inconvenient. If you lose or damage your board (e.g., by dropping it), then we may ask you to pay to replace the board. It is fairly straight forward to "brick" the board by overwriting the contents of its on-board flash, the OS image on the SD Card, etc.; this is discouraged. If this appears to have occurred, please contact the module instructor for assistance. A small number of replacement boards are available if required.

It is easy to replace the SD Card in the board – should it appear to have become corrupted or entered an unusable state (they appear to be sensitive to power being removed from the device while an operation is in progress), let us know and we can provide a replacement. Note that any customisation to your board (e.g., SSH keys, `/data` partition) will not be present on the replacement card – do ensure that you keep the primary copy of any important data on your workstation.

Getting started

The BeagleBone Black will arrive in a compact plastic case, and with a pre-initialised SD Card holding the OS image. For the purposes of this course, you should not need to disassemble the case or eject the card, except to install an image upgrade that might be provided by the instructors. We will power and communicate with the board via a single USB cable. These directions assume that you will be working on an ACS Workstation running Ubuntu Linux. However, our teaching setup has also been used with varying degrees of success with other systems, including FreeBSD and Mac OS X[1]. The BBB appears to the workstation operating system as two USB devices: a USB serial port, which reaches a console, and a USB Ethernet device, which can be used to SSH into the OS instance on the board.

When you plug in the BeagleBone Black, at least one blue LED should light up on the board. It can take a minute or so for the OS image to boot. Once booted, two USB devices will become visible on the workstatation. You can then connect to the console using the following command:

```
$ minicom -o -D /dev/ttyACM0 115200
```

You should be able to log in as `root` without a password. In order to stream data or copy files to and from the board, it is preferable to login to the board using SSH. To do this, you will first need download a **insecure** private key:

```
https://www.cl.cam.ac.uk/teaching/1718/L41/private/2017-2018-l41-insecure-private-key
```

Once downloaded, copy the file to you `~/.ssh` directory on your workstation (ensuring that the correct permissions `chmod 0600 2017-2018-l41-insecure-private-key` are set). To simplify key management your may wish to add the following configuration into your SSH config file (`~/.ssh/config`):

```
Host 192.168.141.100
    IdentityFile ~/.ssh/2017-2018-l41-insecure-private-key
```

Once the key is installed, you should be able to login to the BBB through SSH:

```
$ ssh root@192.168.141.100
```

⚠ **Important!**

The key pair used to provide SSH access to the BBB are the moral equivalent of a default password known to the world. **DO NOT** use this key pair for any other purpose.

⚠ **Important!**

When running as `root`, you should exercise caution: accidentally deleting system files may (will) render your board unbootable or cause loss of your scripts or data.

Now, run your first DTrace script:

```
# cd /data
# dtrace -qn 'BEGIN { printf("Hello world"); exit(0); }'
```

To write the output of a script to a file, you can redirect standard output to a file:

```
# dtrace -qn 'BEGIN { printf("Hello world"); exit(0); }' > data.out
```

[1]We have had mixed experience with Mac OS X and Windows, as the OS image on the BBB implements 'USB target mode' – i.e., appears to be a set of USB devices – and device drivers on operating systems vary quite a bit. If you appear to have reliability problems with the serial console or Ethernet parts, it may well be a device-driver bug. You are welcome to help us debug the problem, but it may be simpler to work with the systems that we have tested with.

We have configured the root filesystem to be read-only by default, in order to reduce the chance of accidents, but also to reduce wear on the SD Card. The `/data` filesystem is writable, however, and is a good place to stick your scripts, data, compiler output, etc., so that it will persist across reboots. You will likely wish to copy files from the BeagleBone Black to your workstation for analysis – e.g., to load it into a spreadsheet. You may also want to edit scripts and source files on the workstation. You can copy files back and forth using `scp`; for example, this command can be run on the ACS Workstation to copy `data.out` to your current working directory:

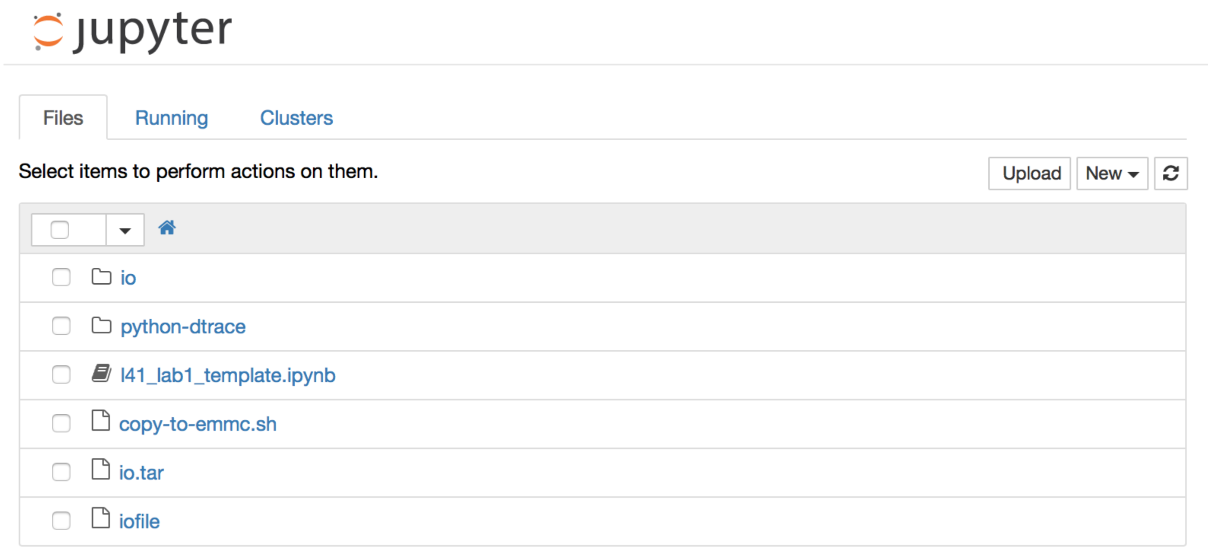
```
$ scp root@192.168.141.100:/data/data.out .
```

Running the Jupyter notebook

To start the Jupyter Notebook `ssh` to the BeagleBone Black (`ssh root@192.168.141.100`) and run the `jupyter` command as follows (note that the `jupyter` command must be run as `root` so that the notebook has permission to execute DTrace scripts):

```
# cd /data
# jupyter notebook
```

On starting, the Jupyter Notebook prints the address on which it can be accessed with a Web browser; this should be
`http://192.168.141.100:8888` (this can take several seconds to appear so please be patient). From the ACS Workstation open a Web browser at this address. The Jupyter dashboard should then be shown:

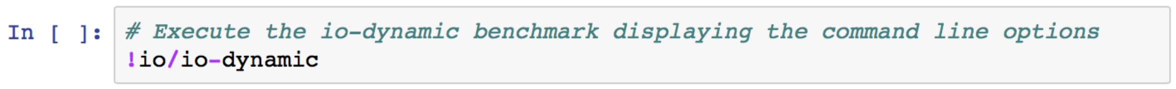


```
io
python-dtrace
l41_lab1_template.ipynb
copy-to-emmc.sh
io.tar
iofile
```

Jupyter (formerly IPython) Notebooks are identified by the file extension `.ipynb`. To open a Notebook simply click on its name. The selected Notebook will open in a separate tab from which it can be edited and run.

Running the benchmarks

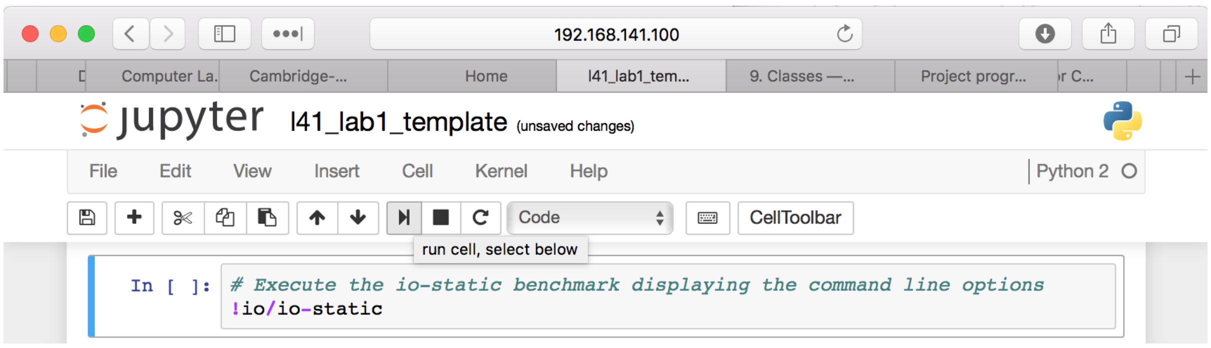
Within a Jupyter notebook (residing in the BBB's `/data` directory) the I/O benchmarks (`io-static` and `io-dynamic`) can be run as follows[2]:



```
In [ ]: # Execute the io-dynamic benchmark displaying the command line options
        !io/io-dynamic
```

[2]Note `!` is a Jupyter "magic" for executing running shell commands.

To execute a cell (in this instance to run the benchmark), simply select it and then press the "run cell" button in the Jupyter Notebook toolbar:



```
In [ ]: # Execute the io-static benchmark displaying the command line options
        !io/io-static
```

The Juypter Notebooks used for the L41 laboratories use a Python kernel running on the BBB. This allows Python scripts and Jupyter shell commands (such as those used to invoke the benchmarks) to be freely intermixed. This allows, for example, running the benchmark in a loop where each iteration uses a different set of parameters:

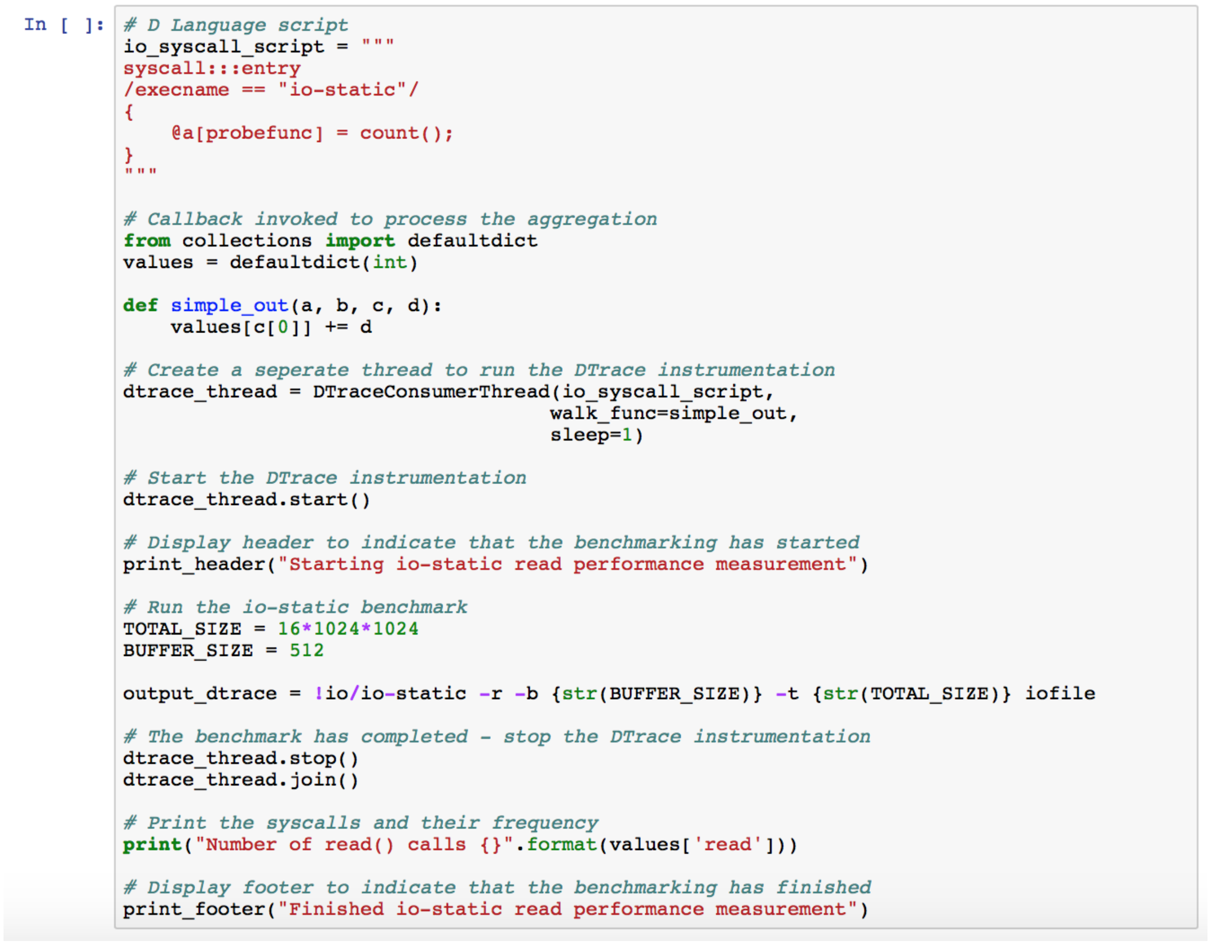
```
In [ ]: # Buffer sizes to compute the performance with
        BUFFER_SIZES = [512 * 2 ** exp for exp in range(0, 16)]

        # Total size of iofile (default size) = 16MiB
        TOTAL_SIZE = BUFFER_SIZES[-1] #16*1024*1024

        for buffer_size in BUFFER_SIZES:
            # Run the io-static benchmark
            !io/io-static -r -B -d -q -b {str(buffer_size)} -t {str(TOTAL_SIZE)}
```

The output of a shell command can be captured into a Python variable (an array with an entry for each row of the output) as follows:

```
In [29]: output = !io/io-static -v -r iofile

         print output[1].split(":")

         ['  buffersize', ' 16384']
```

Running DTrace scripts

DTrace scripts can be executed within the Jupyter notebook with the assistance of the `python-dtrace` module. In order to run the I/O benchmark and also instrument it with DTrace, scripts are executed within a separate thread by instantiating a `DTraceConsumerThread` object. The `DTraceConsumerThread` constructor takes a Python string specifying D-Language script to execute, as well as several other optional parameters. In the example shown below, the script uses a DTrace aggregation to count the number of sycalls invoked by the I/O benchmark. The `DTraceConsumerThread` instance is constructed with the `walk_func` parameter. This callback is invoked periodically with the current value of the aggregation reported by the kernel; in this example the callback stores the total value of the aggregation within a Python dictionary.

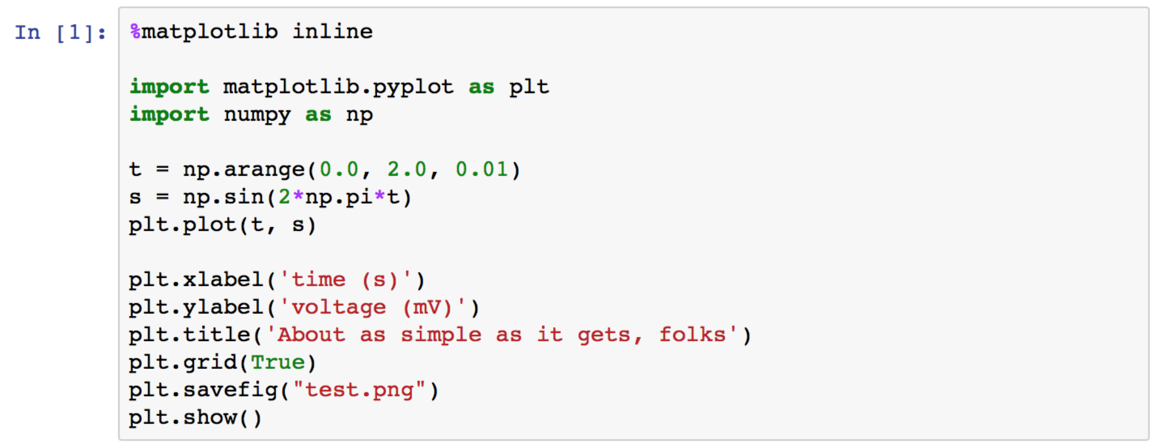


```
In [ ]: # D Language script
        io_syscall_script = """
        syscall:::entry
        /execname == "io-static"/
        {
            @a[probefunc] = count();
        }
        """

        # Callback invoked to process the aggregation
        from collections import defaultdict
        values = defaultdict(int)

        def simple_out(a, b, c, d):
            values[c[0]] += d

        # Create a seperate thread to run the DTrace instrumentation
        dtrace_thread = DTraceConsumerThread(io_syscall_script,
                                             walk_func=simple_out,
                                             sleep=1)

        # Start the DTrace instrumentation
        dtrace_thread.start()

        # Display header to indicate that the benchmarking has started
        print_header("Starting io-static read performance measurement")

        # Run the io-static benchmark
        TOTAL_SIZE = 16*1024*1024
        BUFFER_SIZE = 512

        output_dtrace = !io/io-static -r -b {str(BUFFER_SIZE)} -t {str(TOTAL_SIZE)} iofile

        # The benchmark has completed - stop the DTrace instrumentation
        dtrace_thread.stop()
        dtrace_thread.join()

        # Print the syscalls and their frequency
        print("Number of read() calls {}".format(values['read']))

        # Display footer to indicate that the benchmarking has finished
        print_footer("Finished io-static read performance measurement")
```

Similarly, a `DTraceConsumerThread` can be constructed specifying the `out_func` parameter. This specifies a callback that is invoked by DTrace when a `print`-like action is processed. The Jupyter notebook template for first laboratory provides a example of how this may be used.

Plotting performance measurements

Performance measurements can be plotted inside the Jupyter Notebook using the built-in "magic": `%matplotlib inline`. A simple example, plotting a sine waveform, is given below:



```
In [1]: %matplotlib inline

        import matplotlib.pyplot as plt
        import numpy as np

        t = np.arange(0.0, 2.0, 0.01)
        s = np.sin(2*np.pi*t)
        plt.plot(t, s)

        plt.xlabel('time (s)')
        plt.ylabel('voltage (mV)')
        plt.title('About as simple as it gets, folks')
        plt.grid(True)
        plt.savefig("test.png")
        plt.show()
```

In addition to displaying inline, matplotlib graphs can be saved to the BBB filesystem using the `plt.savefig()` function. Saved files can then be copied from the BBB to the ACS Workstation using `scp` for inclusion in your laboratory reports:

```
$ scp root@192.168.141.100:/data/test.png .
```

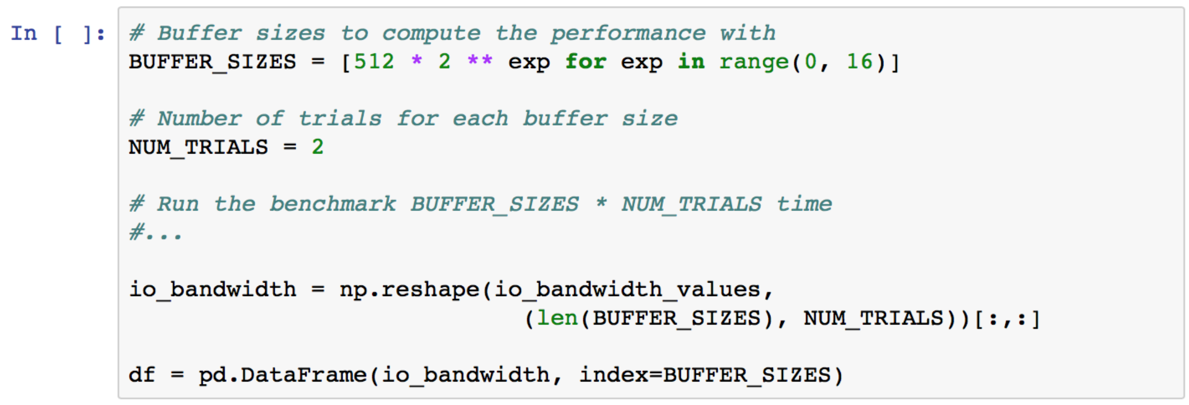
Further information about matplotlib can be found at the project's website: `matplotlib.org`.

Analysing performance measurements

> pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language. (`pandas.pydata.org`)

In this laboratory, performance measurements capture how the *dependent variable* (I/O read and write bandwidth) changes as a result to changes in the *independent variable* (for example, buffer size). This data is best represented in pandas using a `DataFrame`. The pandas `DataFrame` is a tabular structure comprised of rows and columns. A `DataFrame` can be used to, for example, store the performance of a benchmark with the rows representing runs of the benchmark for different buffer sizes. And the columns of the `DataFrame` representing repeated runs of the benchmark for a given buffer size.

A flat list a performance values can be converted into a n-by-m array for loading into a `DataFrame` using the Python module `numpy`'s reshape function:



```
In [ ]: # Buffer sizes to compute the performance with
        BUFFER_SIZES = [512 * 2 ** exp for exp in range(0, 16)]

        # Number of trials for each buffer size
        NUM_TRIALS = 2

        # Run the benchmark BUFFER_SIZES * NUM_TRIALS time
        #...

        io_bandwidth = np.reshape(io_bandwidth_values,
                                  (len(BUFFER_SIZES), NUM_TRIALS))[:,:]

        df = pd.DataFrame(io_bandwidth, index=BUFFER_SIZES)
```

Once a `DataFrame` has been created, statistics such as the median or the 25 and 75 percentile values can be computed directly:

```
df.median(1) # Median value of rows
df.quartile([.25, .75], axis=1) # 25th and 75th quartile value of rows
```

Further information about pandas can be found at the project's website: `pandas.pydata.org`.

Troubleshooting

In the event of the Jupyter Notebook behaving erratically the first port of call is to simply stop the currently executing cell (this can be done from the Jupyter Notebook toolbar). For more serious problems (such as systemic unresponsiveness) the executing "kernel" can be reset from the Jupyter toolbar. This resets the Python runtime's state and is an effective for most problems. If resetting the kernel does not resolve the issue it may be to terminate and restart the `jupyter` command.

As a fallback, it is acceptable to complete the laboratory without using Jupyter Notebooks. Instructions for completing the laboratories using only tools available from the shell can be found in the 2015-16 interation of the course materials:

`https://www.cl.cam.ac.uk/teaching/1516/L41/materials.html`

Shutting down

Simply powering off the BeagleBone Black (e.g., by unplugging it) will be harmless in most situations, but could lead to data loss. It is preferable to instead first run `shutdown -p now` to initiate a software poweroff. Once the system has shut down, the blue LED will turn off, and it is safe to unplug the board.